FreeBSD™ JOURNAL

March/April 2018

BOOK REVIEW

The Phoenix Project

Laptop Shootout

There has never been a better time to run FreeBSD on a Laptop!

Illuminating the Desktop Paradigm

ZFS in 2018 & Onward

Pain & Suffering on the Road to Resume

BORN TO DISRUPT

MODERN. UNIFIED. ENTERPRISE-READY.

INTRODUCING THE TRUENAS® X10, THE MOST COST-EFFECTIVE ENTERPRISE STORAGE ARRAY ON THE MARKET.

Perfectly suited for core-edge configurations and enterprise workloads such as backups, replication, and file sharing.

- ★ **Modern:** Not based on 5-10 year old technology (yes that means you legacy storage vendors)
- ★ **Unified:** Simultaneous SAN/NAS protocols that support multiple block and file workloads
- ★ **Dense:** Up to 120 TB in 2U and 360 TB in 6U
- ★ **Safe:** High Availability option ensures business continuity and avoids downtime
- ★ **Reliable:** Uses OpenZFS to keep data safe
- ★ **Trusted:** Based on FreeNAS, the world's #1 Open Source SDS
- ★ **Enterprise:** 20TB of enterprise-class storage including unlimited instant snapshots and advanced storage optimization for under $10,000

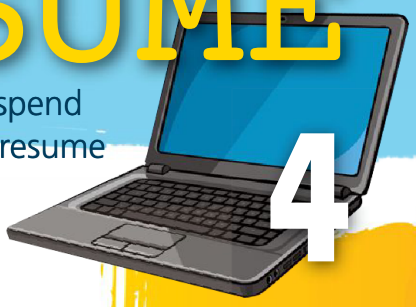
The new TrueNAS X10 marks the birth of a new entry class of enterprise storage. Get the full details at iXsystems.com/TrueNAS.

iXsystems™

Table of Contents

March/April 2018

Pain & Suffering on the Road to RESUME



Despite all the work performed on suspend and resume in FreeBSD, suspend and resume only work on a limited set of systems to date. *By John Baldwin*

FreeBSD Laptop Shootout

A focus on four specific laptop models that offer good support for FreeBSD plus a discussion of resources available to help in picking a laptop for FreeBSD and some of the gotchas in running our favorite OS in a mobile environment. *By George Neville-Neil*

Illuminating the Desktop Paridigm

With a continued focus on modularizing the underlying OS itself, brand new markets can be created that previously seemed impossible due to the shear amount of effort needed to create new "entangled" operating systems from the ground up. *By Ken Moore*

ZFS 2018 & Onward

OpenZFS is continuing to lead the way in file-system development, and the pace is only increasing as more projects and vendors join the effort. *By Allan Jude*

COLUMNS & DEPARTMENTS

Support FreeBSD®

Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.
freebsdfoundation.org/donate

FreeBSD™
FOUNDATION

S&W Publishing LLC
PO BOX 408, BELFAST, MAINE 04915

Publisher • Walter Andrzejewski
walter@freebsdjournal.com

Editor-at-Large • James Maurer
jmaurer@freebsdjournal.com

Copy Editor • Annaliese Jakimides

Art Director • Dianne M. Kischitz
dianne@freebsdjournal.com

Office Administrator • Michael Davis
davism@freebsdjournal.com

Advertising Sales • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).
Published by the FreeBSD Foundation,
5757 Central Ave., Suite 201, Boulder, CO 80301
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsdfoundation.org

LETTER from the Board

Hello and welcome to the March/April issue, where we turn our attention to running FreeBSD on desktops and laptops. We have four main articles this issue: John Baldwin talks about how hard it is to get a suspended system to resume, Ken Moore talks about the Desktop Paradigm, and Allan Jude tells us all about what's coming next in OpenZFS. I got roped into writing an article about running FreeBSD on various modern laptops, which is amusing to me, because it was actually laptops that got me into the BSDs, and FreeBSD in particular.

My first encounter with my own BSD, rather than one shared with a lot of people via a 1980s mini-computer, came in 1993, when I was working at the Universiteit Twente on the eastern side of the Netherlands. My boss at the time was a great fan of Plan 9, AT&T's supposed successor to UNIX, and my research group ran this OS not only on our workstations, DECStation 5000s, which were top of the line at the time, but also on our laptops, which were Compaq Contura's with 4 or 8 megabytes of RAM and 120-megabyte disks. Plan 9 on these laptops was not a pleasant experience—for many reasons—but help was on the way.

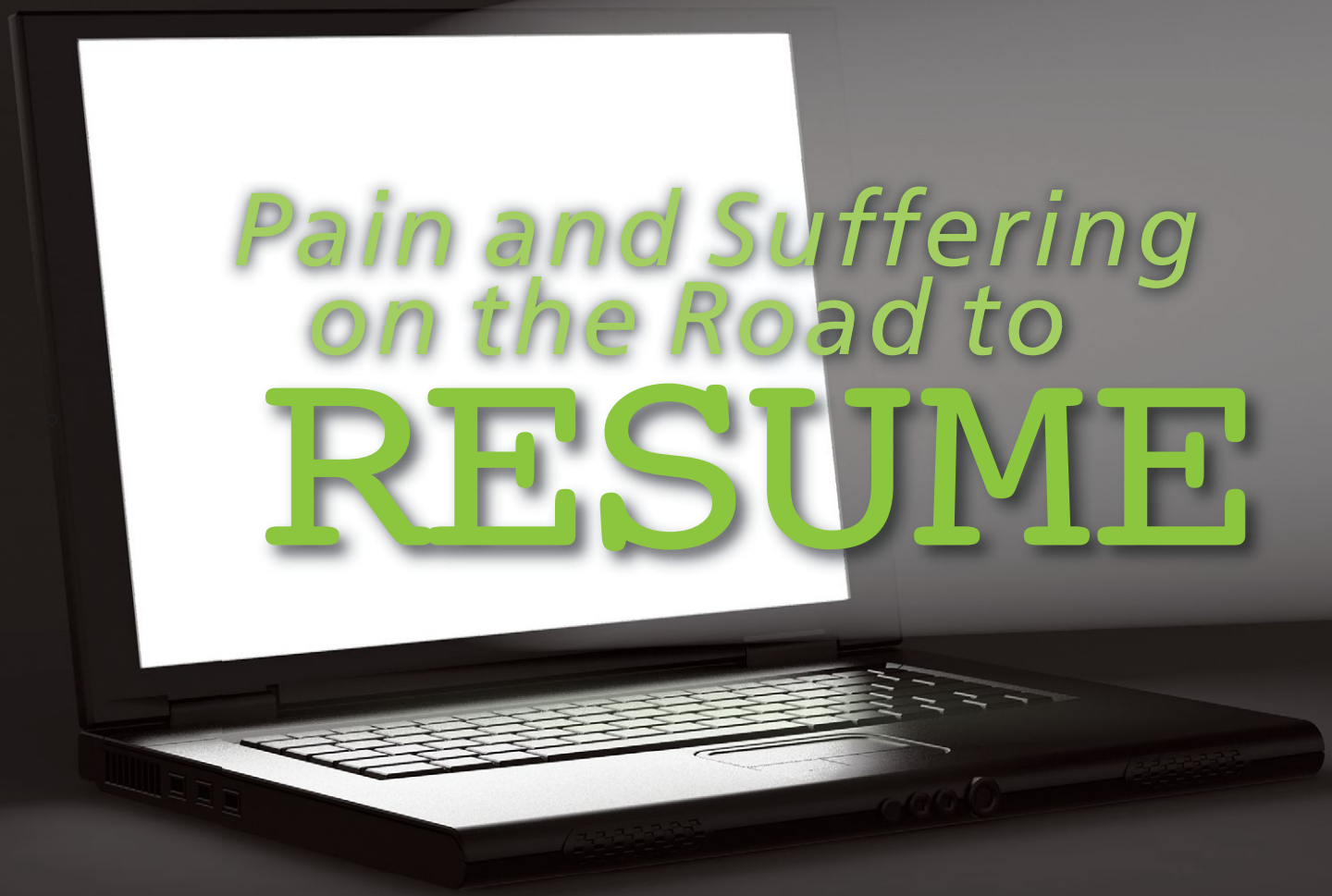
Sometime after April 1993, NetBSD 0.8 was released, and I and one other member of the team flattened our laptops and installed this new BSD-based operating system. Shortly thereafter, the rest of the research team followed suit, leaving only our boss still running Plan 9 on his laptop.

When I returned to the U.S. in 1994, I still wanted to run BSD on a laptop, and by then, FreeBSD 2 had been released. At the time, FreeBSD was the best BSD to run on Intel hardware, as the focus of FreeBSD then was "rock solid performance" on commodity hardware, which meant Intel's processors. I bought a used Winbook laptop from a friend, installed FreeBSD 2.2.2, and have continued to run FreeBSD on laptops to this day, nearly 25 years since that first NetBSD 0.8 installation. And so, as a longtime veteran of FreeBSD on laptops, I can attest to the fact that it has never been a better time to run FreeBSD on laptops.

George Neville-Neil
President of the *FreeBSD Foundation* Board of Directors

SEE TEXT ONLY

BY JOHN BALDWIN



Pain and Suffering on the Road to RESUME

Most consumer computing devices today are mobile devices. These devices are not tethered to a power outlet, but, instead, are able to run on battery power alone. One of the tasks of the operating systems running on mobile devices is to maximize the runtime while on battery. A primary way of achieving this goal is to power off components of mobile devices that are not in use (for example, powering off the screen on a smart phone).

Stock FreeBSD does not currently run on several classes of mobile devices such as phones. However, FreeBSD has run on another class of mobile device for many years: laptops. Traditionally, laptops have not offered very fine-grained power management, but they do permit the entire system to be placed in a low-power state when it is not being actively used. Transitioning a laptop into a low-power state is called "suspend" and returning the system to the fully-operational state is called "resume."

FreeBSD only supports suspend and resume on x86-based systems. In addition, while suspend and resume is supported on both desktop and laptop x86-based systems, development effort in FreeBSD has only focused on laptops. Even then, FreeBSD only successfully suspends and resumes on a subset of x86-based laptops.

Suspend and Resume on x86

Support for suspend and resume in x86-based systems has evolved over time.

The first standard for system-wide power management on x86 is called Advanced Power Management (APM). It supports two different low-power states: standby and suspend. The standby state was able to return to the fully-operational state more quickly than the suspend state. However, the suspend state used less battery power as it turned more internal devices off. While it is possible for the OS to control the power usage of individual devices, this is not required and the BIOS is generally responsible for saving device state when suspending and restoring the saved state when resuming. FreeBSD's APM support relies on the BIOS to manage the power of individual devices during suspend and resume.

APM was replaced by the Advanced Configuration and Power Interface (ACPI).

ACPI is a superset of APM as it includes several other components for managing devices than just power management. ACPI also adopted an expanded set of system sleep states (see Table below).

SLEEP STATE	DESCRIPTION
S0	System is fully operational
S1	No components are powered off
S2	Only CPUs are powered off
S3	All devices other than RAM are powered off
S4	All devices are powered off, state saved
S5	All devices are powered off, state lost

The S1 state in ACPI is similar to APM's standby state, and the S3 state is similar to APM's suspend state. ACPI also includes an S4 state (known as "hibernate") similar to S3 except that the contents of RAM are saved on a hard drive (or similar device). Resuming from S4 requires loading this saved copy back into RAM. Finally, ACPI adds an S5 state which allows the operating system to power off a device. In conjunction with this change, ACPI notifies the operating system when the power button is pressed by the user to give the operating system time to perform an orderly shutdown before the system is powered off.

In contrast to APM, ACPI requires the host operating system to actively manage the power state of many devices in a system during suspend and resume. For example, the OS is required to save and restore PCI config space register values for all PCI devices in the system for sleep states such as S3 that power off devices.

To aid the adoption of hibernation, early systems supporting ACPI's S4 sleep state included an option for the BIOS to assist with saving and restoring the contents of RAM. This option is called S4BIOS. When S4BIOS is supported, the BIOS saves the contents of RAM (usually to a dedicated hard drive partition owned by the BIOS) and restores the contents of RAM during resume from S4. When S4BIOS is not used, the OS is responsible for saving the contents of RAM to some type of OS-managed nonvolatile storage during suspend. During resume, the system powers up and follows the normal bootstrap process. The OS bootstrap is required to recognize a boot as being a resume from S4, locate the saved copy of RAM, and load the saved copy into RAM rather than performing a normal bootstrap.

FreeBSD's ACPI Support

While FreeBSD does include limited support for APM, the majority of modern x86 systems only support ACPI. FreeBSD first supported suspend and resume via ACPI for i386 in FreeBSD 5.0. Support for amd64 (x86_64) was first available in FreeBSD 8.0. The S1 and S3 sleep states are supported, but many laptops only support S3 as S1 offers little power savings. FreeBSD does not support native S4. FreeBSD is able to hibernate on systems supporting S4BIOS, but modern laptops do not support S4BIOS.

System Control Nodes

FreeBSD creates several system control (sysctl(8)) nodes related to suspend and resume under the `hw.acpi` node. Some nodes provide information while other nodes are used to control suspend and resume behavior.

The definition list format is below.

`hw.acpi.supported_sleep_state`
List of ACPI sleep states supported by the host.

`hw.acpi.s4bios`
Indicates if the host supports S4BIOS.

`hw.acpi.sleep_delay`
Number of seconds to pause during a suspend operation after all devices have been suspended, but before the ACPI driver asks the firmware to enter the requested sleep state. Defaults to 1 second.

`hw.acpi.reset_video`
Can be set to 1 to request the kernel to use a legacy BIOS interface to reset the graphics adapter during resume. This generally does not work on modern laptops but did fix issues on some older systems. Defaults to 0 (off).

`hw.acpi.standby_state`
The ACPI sleep state to enter when a userland application requests a transition to the APM "standby" state via the legacy APM interface. Defaults to S1 if the host supports S1.

`hw.acpi.suspend_state`
The ACPI sleep state to enter when a userland application requests a transition to the APM "suspend" state via the legacy APM interface. Defaults to S3 if the host supports S3.

`hw.acpi.lid_switch_state`
The ACPI sleep state to enter when the user closes the lid on a laptop. Defaults to NONE.

`hw.acpi.sleep_button_state`
The ACPI sleep state to enter when the user presses the suspend hotkey on a laptop keyboard. Defaults to the lowest suspend sleep state (S1 - S4) supported by the host.

`hw.acpi.power_button_state`
The ACPI sleep state to enter when the user presses the power button on the host. Defaults to S5.

On a Lenovo ThinkPad X220 the initial values of these nodes after boot are:

```
hw.acpi.supported_sleep_state: S3 S4 S5
hw.acpi.s4bios: 0
hw.acpi.sleep_delay: 1
hw.acpi.reset_video: 0
hw.acpi.standby_state: NONE
hw.acpi.suspend_state: S3
hw.acpi.lid_switch_state: NONE
hw.acpi.sleep_button_state: S3
hw.acpi.power_button_state: S5
```

This indicates that this system supports S3 (suspend), S4 (native hibernate), and S5 (soft-off). Pressing the sleep button suspends via S3. The power button triggers a graceful shutdown and power off. Closing the lid doesn't result in any action. The laptop can be configured to suspend via S3 when the lid is closed by setting the `hw.acpi.lid_switch_state` node to S3:

```
# sysctl hw.acpi.lid_switch_state=S3
hw.acpi.lid_switch_state: NONE -> S3
```

Userland Utilities

FreeBSD provides interfaces for userland utilities to request system suspension or shutdown. Third party applications such as window manager widgets can use these interfaces to permit user-initiated suspensions.

Several base system utilities can also request sleep state transitions. The shutdown(8) and halt(8) utilities accept a `-p` flag to request that the system be powered off via S5 after a clean shutdown. The poweroff(8) utility is an alias for `halt -p`. In addition, the acpiconf(8) utility requests a sleep state from S1 to S4 via the `-s` flag.

Device Driver Support

Device drivers are also responsible for saving and restoring state during suspend and resume. Bus drivers are responsible for saving bus-defined state for each child device before the system is suspended and restoring that state upon resume. Leaf device drivers are responsible for saving and restoring device-specific state. Device drivers should also quiesce active devices when preparing for suspend and restart any paused activity when resuming. Two bus drivers that require explicit suspend and resume support are the ACPI and PCI bus drivers.

The ACPI bus driver primarily manages power states of child devices and power resources. During a suspend request, the ACPI bus driver uses information from ACPI's device tree to place any ACPI devices supporting low power states into a firmware-indicated power state while preparing to suspend. These devices are restored to full power upon resume. In addition, the ACPI device tree describes the relationship between power producers and devices consuming power. If all of the devices that draw from a given power provider are turned off while preparing for suspend, then the ACPI bus driver will turn off the power producer. During resume, the ACPI bus driver restores power to the power producer before any of the associated devices are restored to full power.

Similar to the ACPI bus driver, the PCI bus driver is responsible for placing PCI devices into low power states (using firmware hints to choose specific states when available) while preparing to suspend and then restoring devices to full power on resume. Unlike ACPI, the PCI bus does not define power producer and consumer relationships. If PCI devices depend on discrete power producers, that relationship must be described in ACPI's device tree and managed by the ACPI bus layer. As a result, the ACPI and PCI drivers do share some joint responsibility for power management of PCI devices.

Unlike the ACPI bus driver, the PCI bus driver is also responsible for saving and restoring standardized configuration registers of devices. While preparing to suspend the system, the PCI bus driver takes a snapshot of all of the standard PCI configuration registers including registers to manage resource allocation, interrupt routing, and device control. The PCI bus driver saves the existing values of these registers for each PCI device before the device is placed into a low power state. (Once a device is placed into a low power state it no longer responds to requests to read or write to most configuration registers.) During resume, the PCI bus driver restores the value of these registers after the device has been restored to a fully-powered state.

All bus drivers including both ACPI and PCI invoke two methods in each device's driver to give leaf device drivers a chance to save and restore device-specific state. The `device_suspend` method is invoked by a bus driver while preparing to suspend the system. The `device_resume` method is invoked by a bus driver during resume.

The `device_suspend` method is invoked by the bus driver on each device before placing the device in a low-power state. Device drivers use this method to save a copy of device-specific registers as well as to pause any current activity and disable any active interrupts. For example, a driver for a network interface card will disable the card's receiver and disable any interrupts. In addition, if a device is able to wake the system from suspend, the suspend method should enable this functional-

ity if configured. Some network interface cards permit a system to be awoken via special network packets using a facility known as Wake On LAN (WOL). If the administrator has requested WOL for a specific network card via ifconfig(8), the driver's suspend method should enable WOL.

The `device_resume` method is invoked by the bus driver on each device after the device has been restored to a fully-powered state. Device drivers use this method to restore any device-specific registers and resume any previously-paused activity. Devices supporting wake functionality may also need to disable that feature. For a network interface card driver, a resume method will typically restart transmission of any pending packets and enable the card's receiver.

Debugging Suspend & Resume

Despite all of the work performed on suspend and resume in FreeBSD, suspend and resume only work on a limited set of systems to date. Unfortunately, an issue with a single device driver can cause the suspend and resume of an entire system to fail.

Diagnosing the cause of a suspend or resume failure is rather difficult. Typically, when suspend or resume fails, the laptop just hangs with a powered-off screen. Since the screen is powered off, one cannot use messages displayed on the console to narrow down the cause for the failure. If a laptop includes a non-USB serial port, then one can log messages over the serial port (by using a serial console for example). However, modern laptops generally do not include serial ports. Some laptops do provide a virtual serial port (such as via Intel's AMT), but those serial ports do not reliably resume to working operation during a failed resume attempt in the author's experience.

In some cases, a laptop will mostly resume but leave the screen powered off. The keyboard will still respond to input, however, and if one has suspended in single user mode, one can type blindly to run commands to determine if the laptop is in this state. For example, typing 'poweroff' followed by Enter when in this state should result in a spike in hard drive activity followed by the machine turning off a few seconds later.

This particular failure mode is in fact the most common type of resume failure in the author's experience. If the laptop's graphics adapter is supported by a GPU driver such as i915kms.ko, then loading this driver before suspending will usually resolve this particular failure case.

Additional suggestions for debugging some other suspend and resume failures can be found at https://wiki.freebsd.org/SuspendResume.

Future Directions

There is certainly room for improvement and future work on suspend and resume. Two main areas of work include support for ACPI's S4 state (hibernate) and ACPI's new low-power idle states.

FreeBSD does not yet support native S4 or hibernate. Unlike the S3 suspend mode, resume from S4 follows a normal power-on process prior to resuming the operating system. This leaves the hardware in a state similar to initial boot and depends on the firmware's bootstrap to initialize devices to a known state. This should make resuming from S4 less susceptible to device-specific issues such as failing to restore power to the screen. Hibernate also offers greater power savings compared to S3. While suspended in S3, a laptop still draws power. If the battery is exhausted while in S3, the saved state in memory is lost. With S4, the system does not draw power while suspended and can remain suspended indefinitely. In addition to supporting native S4, FreeBSD should also provide better support for suspend and resume policy management, such as auto-suspending when a laptop's battery is low.

Beyond support for hibernation, the next major challenge for FreeBSD will be support of ACPI's new low-power idle states. ACPI 6.0 introduces a new model for system power management. Rather than using monolithic, system-wide sleep states, low-power idle states encourage more fine-grained power control of both devices and processors. In this model, the operating system seeks to minimize power usage by leaving both devices and processors in low-power states whenever possible. In addition, when responding to wake events such as a WOL packet, the operating system should only restore power to those components of the system needed to handle a specific event leaving other components in a low-power state. While current systems still support ACPI's traditional system-wide sleep states, the expectation is that low-power idle states will eventually supplant system sleep states on systems such as laptops and desktops. This will require a more pervasive understanding of power usage and power producer/consumer relationships through all of FreeBSD's device driver infrastructure as well as alternate approaches to thread scheduling at a minimum. ●

JOHN BALDWIN joined the FreeBSD Project as a committer in 1999. He has worked in several areas of the system, including SMP infrastructure, the network stack, virtual memory, and device driver support. John has served on the Core and Release Engineering teams and organized several FreeBSD developer summits.

There has never been a BETTER TIME to run FreeBSD on a LAPTOP

BY GEORGE NEVILLE-NEIL

While many people in the computing and IT industries still have desktop computer systems, the majority of us who spend our time looking at screens are often looking at laptops. FreeBSD runs well on modern laptops, and in this article, we'll look at a few choices that are particularly well suited to those who want to take FreeBSD with them wherever they go.

Like all open-source operating systems, FreeBSD has had a long, and sometimes difficult, relationship with laptop systems, most often because laptop vendors have been capricious in their selection of parts. The consolidation of most of the major functions of a laptop onto a single processor die has actually improved life for laptop users because it has meant the I/O devices used by the system, including SATA for disk, common video for the display, and networking, on both wired and wireless, are relatively common across a wide range of hardware. A laptop from Dell and one from Lenovo have more in common from the standpoint of their I/O devices now than ever before, and, as a result, the number of device drivers and other pieces of OS software that have to be maintained and tracked over time has shrunk. The various I/O busses have also consolidated, bringing similar economies of scale for the OS developer. All this has translated into more laptops working "off the shelf" with FreeBSD.

For this article, we'll talk about the resources available to help in picking a laptop for FreeBSD as well as some of the gotchas in running our favorite OS in a mobile environment. We'll also talk about four specific laptop models that offer good support for FreeBSD.

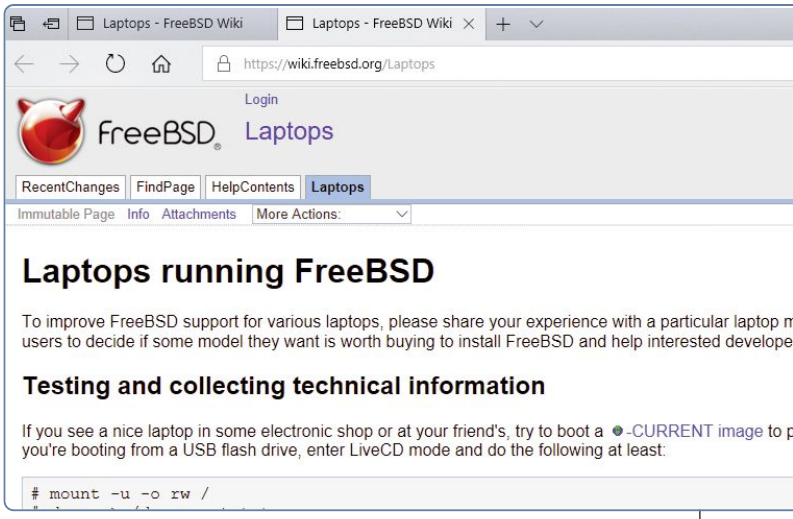
How to Pick a FreeBSD Laptop

Laptops differ from larger systems in a few important ways. They always include a built-in screen and keyboard and they are expected to run on battery power for much of the time they're in use. These three factors—screen, keyboard, and battery life—are very important when selecting a laptop. Another consideration is the type of storage available on the system. Some laptops can be equipped with both an NVME solid-state drive as well as an internal SATA disk, which for FreeBSD users means you can run a ZFS mirror in your laptop, an option we'll discuss later.

Perhaps the best resource for those trying to choose a FreeBSD laptop is the Laptops page on the FreeBSD Wiki (https://wiki.freebsd.org/Laptops). This wiki page is updated and maintained by people who are using FreeBSD on their laptops every day.

The main page of the Laptops wiki is a table of laptops that are currently usable with FreeBSD. The model name in the left column is a link that will take you to detailed information on a particu-

lar laptop model. The main page has columns for Graphics, External DP/HDMI/VGA, Sound, WiFi, Ethernet, USB, Suspend/Resume, Needs config, and Year introduced. Each column contains either a check mark, indicating the feature works out of the box; a blue on white info marker, indicating that some special configuration is necessary; a yellow marker with an exclamation point, which means the feature has not been tested; or, finally, a red mark indicating the feature does not work. The notes column is a set of free form notes that can be used to explain minor nits or other things that do not fit into the columns. Specific instructions on how to get any of the features that need a particu-



Laptop	Graphics	External DP/HDMI/VGA	Sound	WiFi	Ethernet	USB	Suspend / Resume	Needs config	Year introduced
/Dell_XPS13_9360	✔ vt & scfb work in console & X	✔ drm-next	✔	⊗	✔	✔	✔		2017 wifi is ATH10K but field replaceal using intel drm-next drivers, suspend/r HDMI all works. ethernet + HDMI requ additional adapter. wifi support is comi 12.x
/System76 Galago Pro	scfb: ✔		✔		✔	✔			OneDrive
/Thinkpad_T470s	✔	✔	✔	✔	✔	✔	✔	yes	2017
/Thinkpad_X270	✔ drm-next		✔	✔	✔	✔	✔	no	2017Q1

lar configuration to work are contained in the particular model's sub page within the wiki. Note that none of these columns talk about the laptops in terms of comfort, such as how well the pointing device works or the feel of the keyboard. If you're interested in reading a review of how the device feels, you should search the web for reviews written by the press as the wiki is meant as a technical reference. If we look at the sub-page for the Thinkpad X270 (https://wiki.freebsd.org/Laptops/Thinkpad_X270), the laptop on which this article is being written, we find two sections, an Overview, which lists the particular devices present in the laptop, and Notes, which call out any things that ought to be of interest to someone who is trying to use this model.

The four laptops we'll discuss in detail were all released during 2017 and are all well supported by FreeBSD. They are the Dell XPS 13 9360, the System 76 Galago Pro, Thinkpad T470, and Thinkpad X270.

ZFS on Laptops

One of the nice features of FreeBSD is support for the Zetabyte Filesystem (ZFS). Most people think of ZFS as a system for managing large datasets with many terabytes or petabytes, but ZFS has a few features that make it particularly useful to laptop users as well. When ZFS was first imported into the FreeBSD system, laptops were not really powerful enough to run the filesystem, which tends to use a lot of memory. But on a modern laptop, where 16G of RAM is a cheap and easy option, ZFS works quite well. FreeBSD's installer can install your laptop with ZFS by default and this is what we suggest for all laptop users. The two features of ZFS that should be of interest to laptop users are mirroring and snapshots.

If your day-to-day system is your laptop, you probably can't afford to have it be down for long should the disk fail. A ZFS mirrored laptop will allow you to replace either drive, should it fail, without losing any data and with very little interruption to your work flow. Three of the laptops reviewed in this article have the ability to have two storage devices present and used simultaneously. These are the System76 and the two Thinkpads. When using ZFS in a mirrored mode, make sure they get similarly sized disks because the ZFS volume created with a mirror will be the size of the smaller of the two drives. You cannot use a 512G drive to mirror a 1T drive; the resulting storage pool will be only 512G and you'll have wasted half a terabyte.

The Thinkpad models cannot be configured with two drives from the factory; you will have to add the second drive yourself. If you're not the kind of person who spends a Saturday tearing down your laptop, but you still want the second drive, it's best to go with System76, which will give you the option to order one of their systems with two drives installed. If you're going to run ZFS on your laptop, remember to order the maximum amount of RAM, which is 32G on the System76 and 16G on the other three models. Once you install your system, update your sysctl.conf file to give only 4G of RAM to ZFS: vfs.zfs.arc_max=4026259968.

"Choosing a Laptop is Like Picking COMFORTABLE CLOTHES"

Short of a complete loss of a disk, the other day-to-day annoyance is accidental deletion of data. Having ZFS on your laptop means that you have access to easy snapshots, but you have to actually remember to take the snapshots to be able to use them later. By default, ZFS doesn't take the snapshots for you. The zfstools package can help you set up automatic, rolling snapshots, and we strongly recommend you install and set up this package soon after installing your new laptop.

Graphics Support

One of the biggest challenges to open-source operating systems continues to be the secrecy that surrounds the inner workings of various graphics chips. Consolidation in the industry has removed some of the players, and thereby made it somewhat easier for projects run by volunteers to keep up with the shrinking number of graphics chipsets, but the fact that getting reliable information about how graphics chips work or are supposed to be programmed is something that everyone in the field has to put up with. Unlike some other leaders of open-source operating systems, we will not show, graphically, just what we think of these restrictions, but it nevertheless is something you need to consider when choosing a laptop.

When people think of graphics on FreeBSD, they naturally think of Xorg, the most recent incarnation of the X11 system originally developed at MIT in the 1980s. X11 and the Xorg software continue to be the main system for people developing graphical applications on FreeBSD and other open-source UNIX-like operating systems. So, when you choose a laptop, your main concern is whether Xorg runs on the system; if so, how much configuration it requires; and whether or not a driver exists to take advantage of some of the more advanced features of your laptops graphics chips. The Xorg system comes with many different graphics device drivers, including `scfb` that works on nearly any chipset produced in the last 5 years, as well as an even more basic driver called `vesa`. Over the last 18 months, a concerted effort has been made to port and support more advanced drivers that come from the Linux world. The collective name for these more advanced drivers is `drm-next`. Unlike the vesa and scfb drivers, the drm-next drivers support many, if not all, of the features found in the latest chipsets, including acceleration and power-saving modes. While you might think acceleration is only useful for games, this is not the case, because modern graphical user interfaces have elements such as shadows and shine and other useless bling that pretty much require acceleration to work. More importantly to a laptop user are the power-saving features. The screen and graphics are some of the most power-hungry components in a laptop and being able to take advantage of power savings in these subsystems will significantly extend battery life.

One last consideration for graphics drivers is that those in the drm-next collection most often handle suspend and resume correctly. As John Baldwin points out in his article in this issue, getting suspend and resume to work on a laptop is non-trivial, but it is a feature that most of us expect.

Four Laptops for Your Consideration

We all spend a lot of time working on laptops. The following four systems come up time and again as venerable veterans of the FreeBSD road warriors: Dell XPS 13, System76 Galago Pro, Thinkpad T470, and Thinkpad X270. We'll discuss each one and how well it works with FreeBSD. Note we'll be covering technical aspects of how well they work, but not the feel of using them. Choosing a laptop is like picking

comfortable clothes, which means it's best to try them on in person before plunking down one to two thousand dollars to buy one.

Dell XPS 13

The XPS 13 is touted by Dell as a Developer's Laptop, but they don't say which developers they mean. We rather suspect that they don't mean kernel developers, because who in their right might would develop a laptop for them!?

The XPS 13 works well with FreeBSD, supporting the latest graphics drivers from drm-next, which means that the system also reliably suspends and resumes from RAM. The onboard battery gives 8 to 10 hours of power in typical usage, which is a typical working day. So long as you're willing to run a recent version of FreeBSD, something from the CURRENT branch, the built-in wireless works as well.

The configurable options on the XPS 13 are somewhat limited, with a maximum RAM of 16G and four choices for storage, 128G, 256G, 512G, and 1T on PCIe flash disks. The XPS 13 does not have space for a 2.5-inch spinning drive or SSD, so the ZFS mirror trick mentioned earlier will not work. The screen can either be a 1920x1080 or a QuadHD 3200x1800. The model that is referred to on the FreeBSD wiki page is the 1920x1080. One thing to remember about having a lot more pixels is that those pixels draw power, reducing the battery life you might get from the system. The battery in the XPS 13 is of a fixed size and cannot be customized or upgraded, as in other laptops mentioned here.

System76 Galago Pro

System76 is an interesting company specifically dedicated to making laptop systems for open-source software and the only company we've ever seen that specifically allows you to open their system without voiding the warranty.

The Galago Pro is the smallest and lightest of the System76 systems and it can be configured with lots of different options. One of the first that you'll notice is the ability to have both an NVME flash drive and a SATA flash drive, meaning that you can have a laptop with a ZFS mirror. It can be configured with 32G of RAM, which is quite a lot for a laptop. The display is QuadHD (3200x1800) and works well with scfb, unaccelerated Xorg driver. The only real drawback to the Galago is battery life, which stands at a paltry 4 hours in typical usage.

Thinkpad T470

The Thinkpad series has been a stalwart of the open-source community for over a decade. People love these systems so much that Lenovo issued an anniversary edition of one of their systems last year, and a hobbyist group started building modern motherboards to go into old X61 models that haven't been produced in many years. You often see various Thinkpads when you meet up with FreeBSD folks and so we're covering two of their models here.

The T470 is a 14-inch laptop and therefore the largest and heaviest of those we're going to cover. Like the System76, it can be equipped with 32G of RAM as well as two forms of storage but having both NVME and a SATA drive is not an option on the Lenovo website, so you'll have to order the laptop with one type of storage and then add the second one on your own. The cheapest option is likely to get a system with a built-in NVME drive and then buy a 2.5-inch SATA SSD and install it yourself.

The T470 works well with FreeBSD, which is unsurprising because several of the FreeBSD developers use one in their day-to-day work. The accelerated drm-next drivers are supported, and the system correctly suspends and resumes. The system can be purchased with a larger external battery giving the system 14–18 hours of battery life between charges.

Thinkpad X270

The X270, part of the X series, is the smaller cousin to the T470 listed above and is the system on which this article is being written. The X series has a long history with FreeBSD and with many of the project's developers who worked for many years on the X220 and X230 series—some of the first laptops on which accelerated graphics and suspend/resume worked reliably. There continues to be a brisk trade in parts for the earlier X series laptops, since many people are incredibly faithful to them.

The X270, like the T470, works well with FreeBSD, but is smaller and lighter, and if you fly with FreeBSD, you will definitely want to consider this model as more portable as well—possibly usable in an economy seat. The X270 can only be configured with a maximum of 16G of RAM, so if you know you need a lot of memory, you should probably look into the T470 or the Galago Pro. Accelerated graphics from the drm-next driver as well as suspend/resume work as expected. Like the T470, the X270 can be configured with two types

of secondary storage, and, again, you'll have to set this up on your own, and also like its bigger cousin, it can be equipped with a larger external battery. The 72-watt hour battery in addition to the internal 24-watt hour battery give you a whopping 96-watt hours, which, in day-to-day usage, translates to about 12 hours of power.

In Conclusion

If you're a true fan of FreeBSD and want to use it everywhere and as often as possible, this is definitely doable with modern laptops and without compromising on important features such as accelerated graphics, long battery life, and suspend/resume. As an added bonus, you can have an even safer laptop experience if you get two drives and use ZFS mirroring. Following the FreeBSD Laptop wiki page will give you the latest information on which laptops work well with FreeBSD, and, if you happen to purchase a model that's not on the list, please consider contributing your information to the wiki, as this is how you can help our open-source system advance.●

GEORGE V. NEVILLE-NEIL works on networking and operating system code for fun and profit. He also teaches courses on various subjects related to programming. His areas of interest are code spelunking, operating systems, networking and time protocols. He is the coauthor with Marshall Kirk McKusick and Robert N. M. Watson of *The Design and Implementation of the FreeBSD Operating System*. For over 10 years he has been the columnist better known as Kode Vicious. He earned his bachelor's degree in computer science at Northeastern University in Boston, Massachusetts, and is a member of ACM, the Usenix Association, and IEEE. He is an avid bicyclist and traveler and currently lives in New York City.

BSD CERTIFICATION GROUP HAS JOINED WITH LPI.

ONE MORE STEP TOWARDS A

FREE AND OPEN SOURCE WORLD

NOT TO MENTION, JOBS!

Now as a part of LPI, BSD certification will gain a new global reach. It will also benefit as it's relaunched in 2018 to fit into a broader program of free and open source professional credentials. You can help by participating in retooling the certification at lpi.org/bsd.

Linux Professional Institute

COMING IN 2018. WATCH FOR UPDATES.

SEE TEXT ONLY

BY KEN MOORE

illuminating THE DESKTOP PARADIGM

What is a desktop? Is it just the end-goal of a graphical operating system (OS) such as Windows? Or is a desktop the graphical subsystems that can be paired with an operating system to provide application management and control? Is a desktop something that requires keyboard and mouse, or can a smartphone be considered a desktop? These are the kinds of questions that often arise when people discover that I am a desktop developer, but I think the essence of all these questions can be summarized in a single question: "Can a desktop be distinct from the underlying operating system?"

Apple and Android have taken the stance whereby the desktop is just one of several tightly integrated components that together provide a complete graphical interface for the end-user. This approach tends to reduce, or remove, any kind of text-based usage of the OS, greatly limiting the flexibility of that system. On the other end of the spectrum, we have the Unix-like operating systems, which typically have no built-in graphical capabilities and treat all graphical protocols as optional extras. So which approach is the best? (Figure 1)

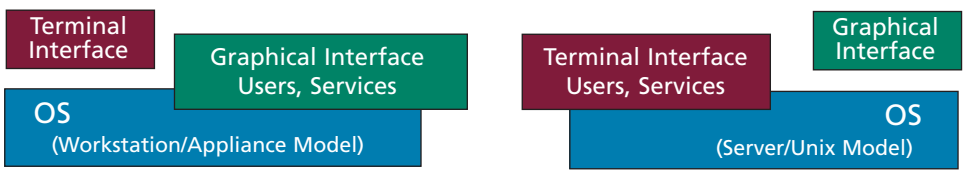
The answer to this OS-model conundrum probably lies in the purpose of the operating system and hardware combination for the end-user. From a distribution standpoint, a smartphone is treated as an appliance and typically allows a very limited set of functions. Because of this, it makes sense that the desktop be inseparable from the rest of the operating system since the desktop is not meant to be modified by the user. The same goes for Apple smartphones and laptops where the software is designed specifically for particular hardware, making the desktop nearly useless without the corresponding appliance. Where the boundaries start to become blurred is when you examine the more general-purpose operating systems, but overall there are two common approaches toward tackling the boundaries on these general-purpose systems.

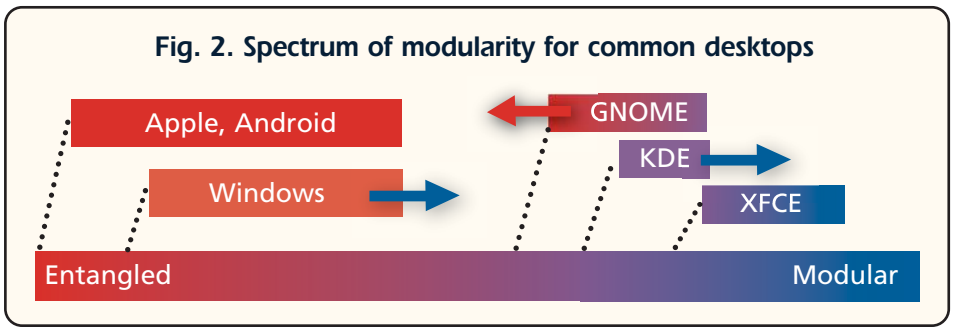
The Windows operating system exemplifies the first approach. It has historically been more like an appliance, where the desktop is almost completely entangled with the operating system. This approach has done very well as a graphical workstation OS for the past couple decades but tends to suffer in the headless-server markets due to the extra overhead for the graphical subsystems. Over time, Windows has been regularly working on untangling the interface from the OS and making the desktop components more modular so that they can enhance their functionality in the server market.

The second approach is exemplified by Unix-like systems. These have completely optional graphical components and correspondingly tend to do better as servers than as workstations. Almost all the desktops that are used with open-source operating systems today, such as KDE, GNOME, and XFCE, were born within this Unix-like environment. However, these desktops have typically striven to copy the Windows model and expect to be treated more like an appliance rather than a modular component of the OS. Figure 2 summarizes the major desktops and where I would place them on the spectrum of desktop modularity.

The design of the Lumina desktop emphasizes the benefits of a fully-modular desktop system and greatly differs from the other open-source desktops. We perceive that a fully modular desktop results in an operating system that can be used

Fig. 1. Common operating system models





for both server or workstation design goals with minimal disruption to the OS, as well as allow the desktop itself to be extremely portable between various operating systems. This is achieved by utilizing a standardized translation layer between the OS and the desktop but suffers from a higher level of code complexity in order to maintain this level of separation. To address this complexity, we have been developing an "OS Interface" class which acts as a completely modular, client-side API object that behaves as a dictionary for common high-level system interactions. By reviewing and categorizing all of the top-level OS interactions that a desktop may need, we are able to easily write and maintain a completely OS-agnostic translation layer that can optionally use OS-specific services and utilities without adding explicit requirements to the desktop itself. Table 1 lists the OS subsystems that Lumina interacts with for optional status indicators and such. Note how few are actually used in this type of desktop format, since the responsibility for OS-modification now falls to the OS, rather than the desktop, and can be as specific to the system as desired.

This type of arrangement results in a highly flexible desktop interface where all the OS interactions are both optional and readily configurable. All that is needed on the OS side to turn a traditional command line server into a graphical workstation is to install a few "desktop" packages such as standalone graphical tools for configuring the OS and the Lumina desktop for everything else. When paired with the flexible interface system that Lumina has also implemented (another subject for another article), a single operating system can power nearly all hardware configurations that utilize a graphical display.

The computing world is changing. The number of types of computer systems, both hardware and software, has been rapidly expanding, and operating systems must become more flexible in order to remain relevant. In my opinion, the BSD operating systems are particularly suited for this metamorphosis because they were designed with a minimalist but self-contained framework for the operating system and need relatively few adaptations to be converted between different types of end-user systems. The Lumina desktop is designed to work hand-in-hand with this type of operating system and together can expand into more graphically-focused market segments. With a continued focus on modularizing the underlying OS itself, brand new markets can be created that previously seemed impossible due to the shear amount of effort needed to create new "entangled" operating systems from the ground up. ●

KEN MOORE is the principal architect of the Lumina desktop and a software engineer at iXsystems Inc.

OS Subsystem	Read	Write	Optional External Tool
Batteries	Existence, status		
Audio	Current volume	Set volume	Full Audio Mixer
Network	Type from device name		Network Manager
External Media	Media shortcuts		
System Updates	Existence, status, logs	Start updates (on logout)	
System Power	Has permission	Start shutdown/reboot	
Screen Brightness	Existence, status	Set brightness	
CPU State	Supported, I/O		
Memory	Supported, status		
Hard Drive Status	Supported, I/O, % full		
Applications			App Store

Table 1: OS Interface interactions for the Lumina desktop

SEE TEXT ONLY

ZFS 2018 and Onward

BY ALLAN JUDE

2018 is going to be a big year for not only FreeBSD, but for OpenZFS as well. OpenZFS is the collaboration of developers from IllumOS, FreeBSD, Linux, Mac OS X, many industry vendors, and a number of other projects to maintain and advance the open-source version of Sun's ZFS filesystem.

Improved Pool Import • 2018Q1

A new patch set changes the way pools are imported, such that they depend less on configuration data from the system. Instead, the possibly outdated configuration from the system is used to find the pool, and partially open it read-only. Then the correct on-disk configuration is loaded. This reduces the number of errors that may prevent a pool from importing. The patch also improves the error messages to better explain what has gone wrong when the pool could not be imported. Additionally, it adds a long-sought-after feature, the ability to import a pool with a missing top-level device. Such imports are read-only since too much data is missing for the pool to be used normally, but some data may be recoverable.

ZFS Device Evacuation • 2018Q1

The ability to remove excess VDEVs from a pool. Data residing on stripe (non-redundant) or mirror VDEVs can be "evacuated", and moved to devices that will remain, allowing the selected VDEV to be removed from the pool. This is accomplished using an indirection table, where ranges of blocks from the removed device are remapped to another device. This feature does not work with RAID-Z VDEVs, and support is not currently planned.

ZFS RAID-Z Expansion • 2018Q4

The ability to add an additional disk to an existing RAID-Z VDEV to grow its size without changing its redundancy level. For example, a RAID-Z2 consisting of 6 disks could be expanded to contain 7 disks, increasing the available space. This is accomplished by reflowing the data across the disks, so as to not change an individual block's offset from the start of the VDEV. The new disk will appear as a new column on the right, and the data will be relocated to maintain the offset within the VDEV. Similar to reflowing a paragraph by making it wider, without changing the order of the words. In the end this means all of the new space shows up at the end of the disk, without any fragmentation. Sadly, existing fragmentation within the VDEV is maintained. In the future it may be possible to use a similar scheme to increase the redundancy level (convert a RAID-Z1 to a RAID-Z2 by adding an additional disk), but that is not part of the current project specification.

ZFS ZStandard Compression • 2018Q2

This project will incorporate Facebook's new Zstandard compression algorithm into ZFS as an optional transparent compression algorithm. This new algorithm is designed to provide compression ratios as good or better than gzip, but many times faster. Developed by Yann Collet, the author of LZ4, the algorithm that has been the default in OpenZFS for many years, ZSTD provides a more enticing trade-off between compression and performance. While not as fast as LZ4, it can achieve much greater compression, and still saturate many spinning disks with just a few CPU cores. It also offers greater control;

with 19 levels of compression to choose from, each dataset can be configured with the optimal amount of compression.

ZFS Adaptive Compression • Under Investigation

The adaptive compression feature is still undergoing preliminary investigation, but, if implemented, would see ZFS automatically adjust the compression ratio up and down as the data is being written, depending on the amount of dirty data waiting to be compressed and written. When the system is not busy, additional CPU time can be allocated to compressing data, but when the throughput of the compression is not able to keep up with the demand of writes, the compression level will be lowered to avoid becoming a bottleneck.

FreeBSD ZFS Spare and Fault Management • 2019Q1

Enhancements to the way FreeBSD boots from ZFS, to bring it more in line with the original design of ZFS. These changes will obviate the need for a freebsd-boot partition and allow the creation of the partition table to be handled by ZFS instead of GEOM, and, therefore, make it possible for zfsd and the ZFS fault management framework to automatically attach replacement devices to the pool and begin the resilver operation without requiring administrator action. Currently this is only possible if the entire device is dedicated to ZFS and does not have a partition table. To be able to boot from the device, a freebsd-boot or EFI ESP partition must exist for the system to boot from. In the original design of ZFS, there is an area dedicated to storing legacy BIOS bootcode, but it is only used under FreeBSD if the disk is partitioned MBR. In other implementations of ZFS, if the entire disk is used for ZFS, a basic partition table is created that marks the entire device as a single ZFS partition. When ZFS was ported to FreeBSD, the 'whole_disk' flag was taken literally, and the raw disk is used by ZFS. With some enhancements that have been proposed upstream to create an EFI partition as part of the ZFS whole_disk layout, with some minor changes FreeBSD could work the same way. This means that hot spares and disks that are swapped after a failure could automatically be labeled, partitioned, and made active in the pool, without requiring a human as they often do now.

ZFS Persistent L2ARC • 2018Q3

The ZFS L2ARC provides a second-level cache, allowing a high-speed device like an SSD or NVMe to provide another tier of cache between the ARC (in RAM) and the main storage pool. This can be extremely important in order to get the required level of performance where it is not economical, or possible, to have an amount of main memory larger than the working set. The L2ARC relies on headers in the ARC to point to the data stored in the L2ARC. The ARC is not persistent, since it is stored in main memory, so when the system is rebooted, the L2ARC contents are orphaned. At boot the L2ARC is considered empty and is refilled as the cache heats up. This can result in a significant loss of performance until the cache has rewarmed. This is somewhat mitigated by configuration options that increase the fill speed of the L2ARC when it is cold. The new Persistent L2ARC feature keeps log records on the L2ARC that can be reloaded after the system boots. Once the system is online, it asynchronously recreates the ARC headers that point to the data on the L2ARC, allowing its contents to persist through a reboot. While they are not available immediately, the system reaches a hot cache state a lot more quickly than without this feature, and with less wear on the L2ARC devices.

ZFS Sequential Resilver • Pending Integration

One of the advantages of ZFS is that because the filesystem and the volume manager are combined, ZFS is aware of which bytes on the disk are in use, and which are not. A disk that is only 1/3 full will only need to resilver that 1/3 of the data, rather than the entire contents of the disk like with a typical hardware RAID. However, to affect this advantage, ZFS scans the content in the order the objects appear in its metadata. This can result in a very large number of random reads and writes, which perform much worse than sequential reads and writes. This enhancement to the resilvering process will scan the metadata and build a range tree of blocks that will need to be resilvered. Once this tree reaches a configured size limit, the largest contiguous range of blocks in the tree are resilvered, and then the metadata scan continues, until the range tree is full again. This approach ensures that the resilver I/Os are done

in large contiguous ranges which will provide much better performance.

ZFS Resilver Prefetch Improvements • Design Review

This set of improvements aims to increase the resilver performance in a different way that complements the sequential resilver work. The new prefetcher is closer to a depth-first search, rather than only working ahead of the scrub and stalling at the end of each sub-tree. In the new system, a demand read completion triggers the next batch of prefetch operations, keeping the I/O queue full. Configuration prevents more than two scrub prefetch I/Os outstanding at once, preventing the prefetch from delaying actual scrub operations.

ZFS Ashift Policy • Design Phase

The goal of this work is to support time-variable geometric. Allowing older 512-byte sector disks to be replaced with newer 4096 byte sector disks without the performance penalty of doing sub-sector writes. Even now, many disks are 4Kn (4k Native), and will refuse to perform sub-sector I/Os. This feature allows an allocation policy to be set that all future allocations will be at least 4k and avoid the performance hit.

ZFS Spacemap Log • Design Review

ZFS uses a data structure called a spacemap to track which space on the disks is free for future allocations. There is both an in-memory and an on-disk representation of the spacemap. Usually a small number of spacemaps are kept loaded at a time, and allocations are made from those maps. Under heavy fragmentation, the system can spend a lot of time trying to find free space to allocate from. The existing spacemap histogram feature helps mitigate this. The spacemap log feature will improve allocation performance by writing allocations and frees to an append-only log, which will then be coalesced into the traditional spacemap data structure once the log exceeds a configured size. In the event of a crash, the last spacemap is loaded, and then the changes specified in the spacemap log and replayed to bring it up to date.

Native Data and Metadata Encryption • Code Review

The native encryption support is based on a design similar to, but not compatible with, that used in the Oracle proprietary version of ZFS. The implementation allows a number of useful features, including authenticated encryption, meaning that not only is the privacy of the data protected, but the integrity as well. The OpenZFS implementation allows each dataset to be encrypted with a different key, or to inherit the key from its parent dataset. Keys can be loaded and unloaded as needed, so data can actually be put at rest, where encryption provides more meaningful protections, by unmounting the dataset and unloading its encryption key. Another useful feature is the fact that the checksum is split between the traditional checksum of the plaintext and the authentication data from the encryption cipher, which protects the ciphertext. This means that ZFS can detect corruption or modification of either form, but it also means that scrub and resilver operations can proceed even when the encryption keys are not loaded. And it means that encrypted datasets can be replicated in their encrypted form, making it impossible to read them on the receiving side without the correct keys.

Windows Port • Early Preview

This last item is mostly just for fun. Some of the people behind the OpenZFS-on-OS-X project wondered how much work it would take to get OpenZFS running on Microsoft Windows. As it turns out, it is not as impossible as you might think. You can check out the GitHub page https://github.com/openzfsonwindows/ZFSin if you want to learn more.

• Conclusions

OpenZFS has come a long way since its split with OpenSolaris in 2010, and the official formation of the OpenZFS organization in 2013. More than 50% of the code that exists in OpenZFS today is either new or replaced from the original OpenSolaris code. OpenZFS is continuing to lead the way in filesystem development, and the pace is only increasing as more projects and vendors join the effort.

You can find out more about many of these and other recently added and upcoming features of OpenZFS from the OpenZFS Developers Summit 2017 wiki page, which includes slides and video from each of the presentations: http://open-zfs.org/wiki/OpenZFS_Developer_Summit_2017. ●

ALLAN JUDE is VP of operations at ScaleEngine Inc., a video streaming content distribution network, where he makes extensive use of ZFS on FreeBSD. Allan is a FreeBSD src and doc committer, and was elected to the FreeBSD core team in summer 2016. He is also the host of the weekly video podcast BSDNow.tv (with Benedict Reuschling), and coauthor of *FreeBSD Mastery: ZFS* and *FreeBSD Mastery: Advanced ZFS* with Michael W Lucas.

SEE TEXT ONLY

svn UPDATE

by Steven Kreuzer

In 2014, as a Google Summer of Code project, FreeBSD developers and students worked to design and implement a modular interface for the loader script interpreter—decoupling the interpreter from the loader. After four long years, it has finally been committed! FreeBSD 12 will ship with a brand-new bootloader based around Lua, which will finally make obsolete the faithful Forth loader that most say overstayed its welcome years ago. While most installments of this column tend to highlight development happening across all subsystems, I thought for this installment I would just focus on the bootloader, since I don't think I have ever seen so much activity happening on such a low-level component of the system.

Add Lua as a scripting language to /boot/loader— https://svnweb.freebsd.org/changeset/base/329166

liblua glues the lua runtime into the boot loader. It implements all the runtime routines that lua expects. In addition, it has a few standard 'C' headers that neuter various aspects of the LUA build that are too specific to lua to be in libsa. Many refinements from the original code improve implementation and the number of included lua libraries. Use int64_t for lua_Number. Have "/boot/lua" be the default module path. Numerous cleanups from the original GSoC project, including hacking libsa, allow lua to be built with only one change outside luaconf.h. Add the final bit of lua glue to bring in liblua, and plug into the multiple interpreter framework, previously committed.

Presently, this is an experimental option. One must opt-in to using this by defining WITH_LOADER_LUA and WITHOUT_FORTH. It's been lightly tested, so keep a backup copy of your old loader handy.

The menu code, coming in the next commit, hasn't been exhaustively tested. A LUA bootloader is 60k larger than a FORTH loader, which is 80k larger than a no-interpreter loader. Subtle changes in size may tip things past some subtle limit (the binary is ~430k now when built with LUA). A future version may offer coexistence.

Bump FreeBSD version to 1200058 to mark the milestone.

Add the Lua scripts from the Lua-bootloader SoC— https://svnweb.freebsd.org/changeset/base/329167

These are the .lua files from Pedro Souza's 2014 Summer of Code project. Rui Paulo, Pedro Arthur, and Wojciech A. Koszek also contributed.

Defer kernel/module loading until boot or menu escape— https://svnweb.freebsd.org/changeset/base/329576

Loading the kernel and modules can be really slow. Loading before the menu draws and every time one changes kernel/boot environment is even more painful. Defer loading until we either boot, auto-boot, or escape to loader prompt. We still need to deal with configuration changes as the boot environment changes, but this is generally much quicker. This commit strips all ELF loading out of config.load/config.reload so that these are purely for configuration. config.loadelf has been created to deal with kernel/module loads. Unloading logic has been ripped out, as we won't need to deal with it in the menu anymore.

Create a "carousel" menu entry type— https://svnweb.freebsd.org/changeset/base/329367

This is a precursor to boot environment support in lualoader. Create a new menu item type, "carousel_entry", that generally provides a callback to get the list of items, a carousel_id, for storing the current value and the standard name/func functions of an entry. The difference between this and a normal menu item, functionally, is that selecting a carousel item will automatically rotate through available items and wrap back at the beginning when the list is exhausted.

Re-work menu skipping bits— https://svnweb.freebsd.org/changeset/base/330020

This is motivated by a desire to reduce heap usage if the menu is being skipped. Currently, the menu module must be loaded regardless of whether it is being skipped or not, which adds a cool ~50-100KB worth of memory usage. Move the menu skip logic out to core (and remove a debug print), then check in loader.lua if we should

be skipping the menu and avoid loading the menu module entirely. This keeps our memory usage below ~115KB for a boot with the menu stripped.

Fix module_path handling with multiple kernels— https://svnweb.freebsd.org/changeset/base/329497

Once we've successfully loaded a kernel, we add its directory to module_path. If we switch kernels with the kernel selector, we again prepend the kernel directory to the current module_path and end up with multiple kernel paths, potentially with mismatched kernel/modules added to module_path. Fix it by caching module_path at load() time and using the cached version whenever we load a new kernel.

Invalidate the screen from menu perspective upon menu exits— https://svnweb.freebsd.org/changeset/base/329986

In the common case, this will effectively do nothing as the menu will get redrawn as we leave submenus, regardless of whether the screen has been marked invalid or not. However, upon escape to the loader prompt, one could do either of the following to reenter the menu system:

— Method 1
```
require('menu').run()
```
— Method 2
```
require('menu').process(menu.default)
```

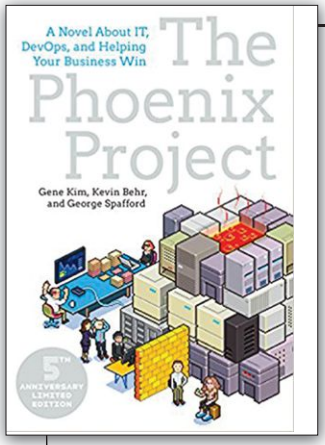
With method 1, the menu will get redrawn anyway, as we do this before autoboot checking upon entry. With method 2, however, the menu will not be redrawn without this invalidation.

Both methods are acceptable for reentering the menu system, although the latter method in the local module for processing new and interesting menus is more expected.

STEVEN KREUZER is a FreeBSD Developer and Unix Systems Administrator with an interest in retro-computing and air-cooled Volkswagens. He lives in Queens, New York, with his wife, daughter, and dog.

SEE TEXT ONLY

BOOKreview by Benedict Reuschling



The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win

by Gene Kim, Kevin Behr, and George Spafford

Publisher .....*IT Revolution Press (3rd Edition, 2018)*
Print List Price *$24*
ISBN.....*9781942788294*
Pages *432*

The Phoenix Project is a novel set in the IT world, written by Gene Kim, Kevin Behr, and George Spafford. Although the characters and the company are fictional, the circumstances and dynamics feel very real. The book opens with the main protagonist, Bill Palmer, being promoted to VP of IT Operations amidst a pile of issues his ousted predecessors left behind.

The book does a great job describing the problems that could happen to pretty much any company. Readers can relate to similar situations they may have experienced themselves and the impacts on their businesses. I was interested in seeing how things turned out from the total mess of the first few chapters. Moreover, I found a lot of parallel experiences from my own work in various organizations and how things are handled (or not handled) in a similar way. Sometimes without knowing any better—as in the book—people do not see any other way of doing things. I think that is one of the interesting things about DevOps and the promise that it holds to make IT work and collaboration with other departments better. DevOps ultimately changes the way things are done on a day-to-day basis.

Companies often see IT as just a cost center or necessary evil that does not provide much. I remember one of our professors reminding us that IT is not our core business, but it is the core of our business. The book describes how IT is at first seen as just a department every other department relies upon, but is causing constant grief and trouble. Over the course of the book, it becomes clear to the people in IT that they need to look beyond the boundaries of their own department. Identifying what other departments like sales and finance need from IT changes priorities and objectives. Making an impact on those departments directly affects business outcomes in positive ways beyond running servers reliably and applying changes to production systems. That is one of the promises of DevOps, that over time it ultimately transforms the whole company into a better, more efficient and productive performer.

Pundits may find some of the approaches described in the book naive to implement in a large corporation or that the results cannot be seen as fast as in the novel. However, the book also has people that are resisting the plans that Bill is implementing and they try to sabotage his efforts. I like how the authors made an effort to transcribe some of the (fictional) emails in the form you would get in your email program (including From:, To:, and Subject: lines). This makes the book more realistic and business-like. Also, the graph about wait time being the relationship of busy time divided by idle time for resources was particularly insightful for me. It explains a lot of things and cautions you to be careful not to overcommit yourself.

What I particularly liked is that the book is not only focusing on the business side of IT operations, but also how it affects the personal life of Bill with his wife and kids. All too often, Bill is being called at home after hours and has to deal with current business problems instead of spending time with his family. Often, working in IT

affects the whole family when things go bad.

Reading this book was a continuation of what I learned in the Agile Software Development course at the university. Things like the Toyota Production System, Lean Manufacturing, and Kanban are mentioned and applied in a concrete business setting in *The Phoenix Project*. Although I wished they would have elaborated more on Kanban and how to use it, I can understand that this is just one tool and a (nevertheless important) means to an end in the bigger picture of DevOps. There are other books on Kanban and this book motivated me to try it out for my own personal projects.

Not everyone reading the book will be in a management position like Bill and cannot change a whole department at once. But these concepts also work for a single person—things like identifying and prioritizing the most important work, mastering your skills by constantly applying and refining them, creating a value stream map and using Kanban to visualize the work, eliminating waste and rework. Of course, some of these concepts require deeper elaboration and tips on how to implement them, which is why there is a resource guide in the Appendix. Additionally, the authors mention *The DevOps Cookbook*, which is now called *The DevOps Handbook*, which is meant to be read after *The Phoenix Project*.

All in all, I can recommend the book to everyone working in IT. The story is interesting enough that non-IT workers can also see what IT is dealing with. And the concepts about work can be universally applied to pretty much any profession. ●

BENEDICT REUSCHLING joined the FreeBSD Project in 2009. After receiving his full documentation commit bit in 2010, he actively began mentoring other people to become FreeBSD committers. He is a proctor for the BSD Certification Group and joined the FreeBSD Foundation in 2015, where he is currently serving as vice president. Benedict has a Master of Science degree in Computer Science and is teaching a UNIX for software developers class at the University of Applied Sciences, Darmstadt, Germany.

SEE TEXT ONLY

by Deb Goodkin

conference REPORT

FOSDEM 2018 / BSDDevRoom / FreeBSD Developer Summit

I recently had the opportunity to attend FOSDEM 2018 (https://fosdem.org/2018/about/), a two-day, free and open-source software conference, held in Brussels, Belgium, on February 2–3. Founded in 2000, this is an annual conference organized and run by volunteers, and it's free! Because it's free, they get a lot of attendees—in fact, over 5,000 attendees this year!

Though FreeBSD has been promoted at this conference over the years, it was my first time attending. I wasn't exactly sure what to expect. I had heard stories that it was overwhelming with lots of people, and to not expect to find lunch, unless you wanted to wait 30+ minutes in a queue in front of one of the lunch trucks. It turns out most of what I had heard was true, but I have to say it was an amazing conference and a perfect opportunity to raise awareness about FreeBSD.

The Foundation sponsored the FreeBSD Developer Summit that preceded the conference, and the FreeBSD stand at the conference. With a two-bag, 70-pound/bag allowance on United, I decided to stuff my bags with as many FreeBSD Journals, handouts, and swag as I could carry. I ended up with two super-heavy bags, that were a little difficult to navigate and lift off the luggage carrousel. But, what's not to like about getting a good workout while traveling?

The Foundation sponsors these types of events to promote FreeBSD and recruit new users and contributors to the Project. We also sponsor developer summits around the world to provide face-to-face opportunities for FreeBSD contributors to work together, share knowledge, and have productive discussions on different areas of the FreeBSD Project.

FreeBSD Developer Summit

The summit was held in the basement of one of the local hotels. Benedict Reuschling organized and ran the event, which had about two dozen attendees. The summit kicked off at 9 a.m., with everyone introducing themselves, including what they were working on, and what they would be interested in discussing during the day.

After all the introductions, Allan Jude talked about the Google Summer of Code (GSoC) program, explaining what a typical project would look like and how we needed more small projects submitted, as well as more volunteer mentors. I brought up the Outreachy program, which is modeled after the GSoC program, though directed to women and minorities. I've wanted to participate in this program for years, and finally believe we are ready to support it. GSoC is so beneficial for the project that I don't want to take resources away from it. Since attending the conference, we decided to apply for the next session in six months.

Matthew Rice and Fabian Thorns from Linux Professional Institute discussed their taking ownership of the BSD certifications, explaining why, and what their plans are for expanding online tutorials. Right now, they offer various levels of certifications for Linux, and other areas such as DevOps. Going forward, they plan to put material together that is common to both operating systems to provide operating system agnostic educational material. They have a BSD advisory group that is headed by Dru Lavigne, who will have oversight into the BSD certification support.

Some of the other topics discussed were DTrace, graphics support, storage, and possible release models. The discussions were productive, and I felt like we accomplished a lot. Many of the attendees continued conversations over dinner at a nearby restaurant. I joined a group of open-source leaders at a dinner sponsored by Google. Connecting with other open-source people outside of FreeBSD is important for finding out what others are doing to improve their projects and communities. It's also an

opportunity to lend our voice to keeping open source a healthy ecosystem.

FOSDEM and the BSD devroom

As I mentioned earlier, FOSDEM is big! There were 651 speakers, 690 events, and 57 tracks, all of which took over 33 rooms spread across several buildings of the Solbosch campus of the Université Libre de Bruxelles. There were keynotes, main tracks, and at least 42 developer rooms, lightning talks, and certification exams including a BSDCG exam.

The BSD devroom, organized by Rodrigo Osorio, took place the first day of FOSDEM. There were 11 talks that day, to a full room of over 100 people. I gave my talk first, *The FreeBSD Foundation and How We Are Changing the World* (https://fosdem.org/2018/schedule/event/the_freebsd_fundation_how_we_can_change_the_world/).

A handful of people wanted to learn more about FreeBSD. That was a concern I had as I prepared—would the room be only BSD people? Or, would there be non-BSD people too? I usually give this talk to highlight the work we are doing to support FreeBSD, so that was my focus. I realized after speaking with someone who expressed an interest in more technical information that we should give an *Introduction to FreeBSD* or *FreeBSD Advantages* talk next year as well as the *Foundation Highlights* talk. It was valuable feedback.

After my talk, a few people asked for my card because they wanted to talk with me about their companies using FreeBSD. One gentleman wanted to discuss whether we were considering offering FreeBSD support, which led to a long discussion.

I sat in a few other talks that I found interesting, including *BSD from scratch— from source to OS with ease on NetBSD* and *pot: another container framework based on jails and ZFS*.

We also had a FreeBSD stand, also known as a table or booth in the U.S. FreeBSD contributors volunteered to staff the stand and talk to people. I showed up at our swamped stand right around lunch time. I quickly understood why so many people asked me if we had a stand this year. You couldn't see our table with all the people in front of it! We all agreed that next year we will need a big sign to help us stand out (no pun intended!). However, it didn't stop hundreds, if not thousands, of people from stopping by our stand. Most of the visitors were familiar with FreeBSD. Many had questions about their setups, and our volunteers were able to help them. I had my Raspberry Pi 3 FreeBSD demo setup on display, which drew a lot of people to check it out. We also had a sign that showcased many of the companies that use FreeBSD. The purpose was to highlight how FreeBSD is growing and that there are many marquee companies successfully using it. Lastly, there were a few people who genuinely wanted to contribute to the Project, which was exciting to hear.

Our stand was next to the illumos stand, which allowed Allan Jude to sit between the two projects, and install bhyve on illumos with UEFI.

I spent the rest of the day either behind, or in front of, our stand. At times, we had five people behind the stand.

Stickers are the commodity there. You must have stickers! At one point, we thought we ran out, but luckily I found more hiding away. The luggage identifiers we gave away didn't last long, and someone suggested we create a man page for them, since we are so good at documentation. I thought that was an excellent idea. Next year, we definitely need to bring at least 1,000 stickers and maybe t-shirts to sell.

The free swag, Groff the BSD Goat, and the RPi 3 demo drew many people to the stand, but most people stayed to talk with us, to either learn more about FreeBSD, or, if they were familiar with FreeBSD, to ask questions.

FOSDEM Day 2

The second day of FOSDEM was pretty much like Day 1, except we had more volunteers at our table. Plus, you could actually walk through the crowd without elbowing people. We also discovered that the lunch truck lines were short if you went early enough! Besides talking to attendees about FreeBSD, I also spent time getting to know some of our community members, some of whom I had met over social media, and others I had never met before. Not only do I enjoy hearing what people are working on, but these conversations also provide a chance to talk about what we can do to help, encourage them to give a talk at a conference or meetup, or write an article about their work. A lot of information exchange happens when you can meet with people from the community face-to-face.

Later in the day, another interesting discussion began when a FreeBSD developer commented that he didn't want people switching to FreeBSD, just because of systemd,

but because of FreeBSD's many positive attributes. I asked him the top three things he liked about FreeBSD. Then two others from the Project joined in the discussion, which made for an informative conversation!

Some of the reasons were:

- DTrace, ZFS, netgraph, and DummyNet
- Interface between base system and packages is well defined
- Base system is a whole system, including kernel, userland, and tools
- Tons of third-party software available in the ports

When the conference was over, everyone stepped up to help clean up. The rule was that we had to leave the space as we had found it. The whole effort took a while, not because there was a lot to pack and clean up, but because we needed to say goodbye to many of the people we had met over the weekend.

The Chocolate Mission

Unknown to the others coming back with me for dinner, I had one more mission to accomplish. I had to get chocolate to bring back for my friends and family. Unfortunately, it was Sunday evening. I didn't think anything would be open, but at least I wanted to try. We left as quickly as we could, ordering two Ubers for the seven of us. I was a little smug, when I ordered mine first. The two groups waited along the busy road, and then my group noticed on the tracker that our Uber wasn't moving. It still wasn't moving when Benedict's group was picked up and smiled and waved at us, as we continued to stand there confused in the bitter cold. Fortunately, the Uber cancelled, and I quickly ordered another one that showed up soon afterward.

We all met up shortly after dropping off our bags at the hotel and headed out to look for chocolate. Finding chocolate stores around Grand Place is like finding health food stores in Boulder. They are everywhere! Fortunately for us, they were all open. The best part is that you get to sample chocolates at all the stores, and then decide which store (or chocolate) you like the best. We all ended up buying tons of chocolate at Allan Jude's favorite store, Corné Port-Royal. It was another social opportunity for our community to have some fun.

One can never have too much Belgian chocolate!

Our community is made up of some truly wonderful people. They like to hang out with each other, whether it's working on a project together, or exploring a new city together. Our group of seven FreeBSD contributors' true colors shined when we went out to dinner the last night. After we were satisfied with our chocolate purchases, we quickly dropped off our loot at our hotel and headed out for dinner. Benedict had found a restaurant earlier that he thought looked good, so we went there to check it out. Unfortunately, we quickly realized there was an issue. One of us was vegan and two were vegetarians. This place had no vegan selections. While our FreeBSD vegan friend suggested that he was fine going off on his own, there was no way we were going to leave him.

Determined to find a vegan-friendly restaurant, we headed off following Google Maps. Sadly, that didn't work out very well. The one place we found was closed. Undeterred, I stepped next door, which happened to be a chocolate shop. I managed to control myself and asked the employee if she knew of any vegan places. Luckily, she recommended one only a few blocks away. Finding that place was not easy, and when we finally found it, they didn't have seating for eight. Did I mention one of the illumos people joined us in our hunt? Yes, we are a friendly group of people! At that point we decided to split up. Three of us stayed at the vegan-friendly café and the rest went off and found a restaurant that they were happy with. Our vegan friend enjoyed his vegan cake! I liked my meal there too, and yes, it was vegan.

Overall, it was a great experience. It provided an excellent opportunity not only to promote FreeBSD, but also to meet others from the community, and people interested in open source. I look forward to promoting FreeBSD at more open-source conferences around the world this year. ●

DEB GOODKIN is the Executive Director of the FreeBSD Foundation. She's thrilled to be in her 13th year at the Foundation and is proud of her hardworking and dedicated team. She spent over 20 years in the data storage industry in engineering development, technical sales, and technical marketing. When not working, you'll find her on her road, gravel, or mountain bikes, training for a running race, hiking with her dogs, or playing with FreeBSD on her various systems.

APRIL–JUNE 2018 BY DRU LAVIGNE

2018 Events Calendar

The following BSD-related conferences will take place during the second quarter of 2018.

ZFS User Conference • April 19 & 20 • Norwalk, CT

http://zfs.datto.com/ • The second annual conference focuses on the deployment, administration, and tuning of the ZFS filesystem. The conference consists of two days of talks and networking. This event requires registration at a nominal fee.

LinuxFest Northwest • April 28 & 29 • Bellingham, WA

https://www.linuxfestnorthwest.org/conferences/lfnw18 • This annual open-source event features presentations for everyone from the novice to the professional. There will be a FreeBSD booth in the Expo area. This event is free to attend.

Rootconf 2018 • May 11 & 12 • Bangalore, India

https://rootconf.in/2018/ • Rootconf is India's principal conference where systems and operations engineers share real-world knowledge about building reliable systems.

BSDCan • June 6–9 • Ottawa, Canada

http://www.bsdcan.org/2018/ • The 15th annual BSDCan provides 2 days of tutorials, a Developer Summit, and 2 days of technical talks that appeal to a wide range of people from extreme novices to advanced developers. Registration is required for this event.